



DAW-IT00217

PR102 - Galería Magic

Índice

[Índice](#)

[Listado Requisitos](#)

[Requisitos no funcionales:](#)

[Requisitos funcionales:](#)

[Planteando el proyecto](#)

[Pautas que van a seguir las Cartas](#)

[Creación Clases](#)

[Atributos](#)

[Métodos](#)

[PlantUML](#)

[Anotaciones importantes](#)

[Creación de Base de Datos](#)

[Modelo Conceptual](#)

[Modelo lógico](#)

[Modelo físico](#)

[Script.sql de la Base de Datos](#)

[Diseño estructura](#)

[index](#)

[CSS](#)

[Javascript](#)

[Relacionando thumbnails con cartas](#)

[Ocultar la carta](#)

[Ordenando nuestras cartas](#)

[Información de la carta](#)

[Formulario](#)

[Estructura](#)

[Tipos de datos y validaciones](#)

[Batería de pruebas](#)

[Ejemplo](#)

[Herramientas utilizadas](#)

[PlantUML](#)

[PHP Debug](#)

[Notion](#)

[Draw.io](#)

[Problemas y soluciones](#)

[Bibliografía](#)

Listado Requisitos

Requisitos no funcionales:

- Orientación a objetos

RWD

- Batería de pruebas automatizadas
- Modelado y diseño de base de datos
- Uso de herramientas de control de versiones para el desarrollo del proyecto
- Publicación y puesta en producción en el subdominio magic.tudominio.aaa.
- Estará debidamente documentado con bloques de documentación entendibles por phpDocumentor
- La documentación del código estará generada y desplegada en producción en la carpeta 02_deephp

Requisitos funcionales:

- Debe tener un pull de base de 10 cartas diferentes.
- Poder dar de alta nuevas cartas a nuestro mazo.
- Saber cuantas cartas en total tenemos
- Saber cuantas cartas diferentes tenemos.
- Para presentar las cartas, se visualizarán con un thumbnail (a modo de galería), mostrándose la imagen de la criatura/artefacto/... y su nombre, al hacer clic se mostrará una ventana modal con la carta Magic.
- Las cartas aparecerán, de primeras, ordenadas según esté establecido en la base de datos, pudiendo el usuario ordenarlas por nombre (asc/desc), por el orden establecido (asc/desc) y tipo (asc/desc), sin recargar la página.

Planteando el proyecto

Nada más recibir el proyecto he leído todos los requisitos tanto funcionales como no funcionales. Una vez leídos estos requisitos como no sabía como funcionaban las cartas Magic realicé una exhaustiva investigación sobre la anatomía de la carta

Anatomía de una carta

La Academia Magic es una columna diseñada para ayudar a los jugadores que acaban de empezar, enseñándoles más cosas sobre el juego y mostrándoles los recursos disponibles en la web para que aprendan más. La columna está escrita en modo lineal, como un libro, por lo que cada lección se basa en lo aprendido

 <https://magic.wizards.com/es/articles/archive/magic-academy/anatom%C3%ADa-de-una-carta-2006-07-18>



y sobre que era una ventana modal y en que se diferencian sobre los pop-ups, pop-over, lightbox etc. La principal diferencia y característica de la ventana modal es que aparece en el centro de la página bloqueando todas las funciones de la web para concentrar la acción en la ventana recién abierta, estas ventanas se activan cuando el usuario hace click en un llamado en nuestro caso y siguiendo los requerimientos del proyecto se activará cuando se haga click al thumbnail de foto y nombre de la carta.

Pautas que van a seguir las Cartas

Después de estudiar exhaustivamente la anatomía de las cartas, nuestras cartas se van a regir bajo las siguientes pautas:

- Cada carta creada va a tener un elemento de los siguientes (Blanco,Azul,Negro,Rojo,Verde).
- Si la carta pertenece al elemento Blanco los iconos de mana de color van a ser obligatoriamente blancos (en mazos modernos hay iconos de mana con diferentes colores, este no es nuestro caso ya que en estos mazos no había estos iconos de varios elementos), si la carta pertenece al elemento azul todos los iconos de mana de color van a ser azules y así sucesivamente.
- Cuando la carta no tenga iconos de mana de color o mana incoloro se creara con 0.
- El mínimo de mana incoloro estará 0 y 9.

- El texto de las cartas de habilidad y de descripción estará limitado a 150 caracteres.
- La fuerza y resistencia podrá contener números del 0 al 9 y/o el carácter *.
- El nombre del autor estará limitado a 30 caracteres.
- El fondo de la carta será del mismo color al elemento que pertenece.

Creación Clases

Atributos

Siguiendo la estructura anterior vemos necesario crear solo una clase llamada carta la cual va a contener los siguientes atributos:

```
private int $id_carta;
private string $nombre;
private string $elemento;
private string $iconomana;
private string $fondo;
private int $nummanacolor;
private int $manaincoloro;
private string $imgcriatura;
private string $tipo;
private string $subtipo;
private string $simboloexp;
private string $habilidad;
private string $descripcion;
private int|string $fuerza;
private int|string $resistencia;
private string $autor;
```

Estos atributos hacen relación a todas las partes que tiene que tener una carta según nos indica la web oficial de Magic The Gathering.

Métodos

Pensando en nuestra aplicación nuestras cartas o thumbnail van a ser y mostrarse en modo galería por lo que no van a tener un comportamiento o acción como puede ser atacar o defender a otras cartas, por este motivo tampoco se crean especializaciones con la misma por tipo de elemento por ejemplo (**este motivo lo desarrollamos ampliamente en el apartado de Anotaciones Importantes que se encuentra a continuación**). Sabiendo esto la única opción que tiene la carta es la de crear un contenedor donde se muestre su previsualización y la carta en una ventana modal al hacer click en la previsualización, por ello nuestra clase carta va a tener un método construct , un método mostrarcarta()(Se realiza en un método nuevo y no en __toString por si hay que utilizar este método mágico en un futuro), unos métodos que van a ir comprobando si los datos introducidos en el construct son válidos y si pasa este filtro se crea el objeto si no no llega a crearse y la habilidad de poder destruirse cuando ya se encuentre lista para ser mostrada en nuestra aplicación.

```
public function __construct($id_carta, $nombre_carta, $color_elemento, $icono_mana, $fondo_carta, $num_mana_color, $num_mana_incoloro,
    $habilidad, $descripcion, $fuerza, $resistencia, $autor);
public function mostrarcarta();
public function comprobarnombre($nombre_carta);
public function comprobarelemento($color_elemento);
public function comprobariconomana($icono_mana);
public function comprobarfondo($fondo_carta);
public function comprobarnummanacolor($num_mana_color);
public function comprobarnummanaincoloro($num_mana_incoloro);
public function comprobardescricion($descripcion);
public function comprobarfuerza($fuerza);
public function comprobarresistencia($resistencia);
public function comprobarautor($autor);
public function __destruct();
```

El método de mostrarcarta() va a tener una serie de instrucciones que se encargarán de imprimir la carta y el thumbnail de manera correcta si existen en nuestra aplicación web.

PlantUML



```

@startuml
skinparam classAttributeIconSize 0

class Carta {

    - int $id_carta
    - string $nombre
    - string $elemento
    - string $iconomanana
    - string $fondo
    - int $nummanacolor
    - int $manaincoloro
    - string $imgcriatura
    - string $tipo
    - string $subtipo
    - string $simboloexp
    - string $habilidad
    - string $descripcion
    - int|string $fuerza
    - int|string $resistencia
    - string $autor

    +__construct($id_carta, $nombre_carta, $color_elemento, $icono_mana, $fondo_carta, $num_mana_color, $num_mana_incoloro, $imagen_carta, $tipo, $subtipo, $icono_expansion, $habilidad, $descripcion, $fuerza, $resistencia, $autor)
    +mostrar()
    +comprobarNombre()
    +comprobarElemento()
    +comprobarIconoManana()
    +comprobarFondo()
    +comprobarNumManacolor()
    +comprobarNumManaincoloro()
    +comprobarImgCriatura()
    +comprobarTipo()
    +comprobarSubtipo()
    +comprobarHabilidad()
    +comprobarDescripcion()
    +comprobarFuerza()
    +comprobarResistencia()
    +comprobarAutor()
}

```

```

+comprobarnombre()
+ comprobarelemento()
+ comprobariconomana()
+ comprobarfondo()
+ comprobarnummanacolor()
+ comprobarnummanaincoloro()
+ comprobarmgcriatura()
+ comprobartipo()
+comprobarsubtipo()
+ comprobariconoexp()
+ comprobarhabilidad()
+ comprobardescripcion()
+ comprobarfuerza()
+ comprobarrresistencia()
+ comprobarautor()
}
@enduml

```

Anotaciones importantes

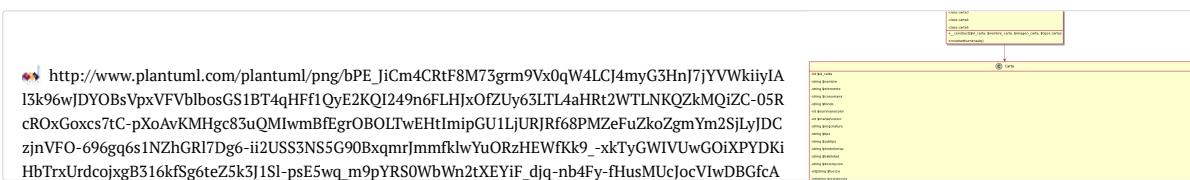
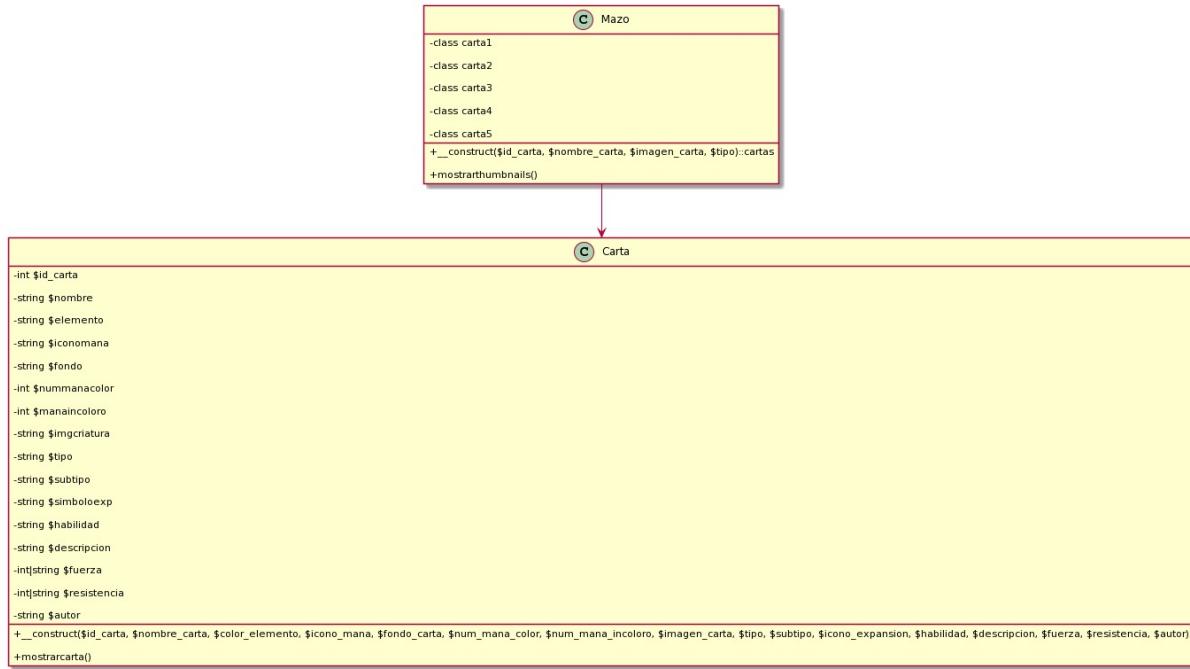
Nota: En POO de la carta Magic tendría sentido crear especializaciones de la misma por color de elemento, en este caso no vamos a utilizar estas especializaciones ya que solo vamos a realizar una galería con las cartas y estas especializaciones no tendrían un uso lógico en este proyecto, en cambio si las cartas atacaran y/o defendieran sería muy buena opción replantear la clase carta y sus especializaciones, (Habría que investigar si el tipo y subtipo de carta influye a la hora de combatir por lo que podría convertirse en otra especialización), en nuestro caso tampoco influye el tipo o subtipo ya que es solo a modo de texto.

Pensando en los métodos que podría tener la clase de carta llegamos a la conclusión que va a tener la "habilidad" de mostrarse, la cual desarrollaremos más adelante ya que involucra la creación de la estructura de la web.

Nota 2: Hoy día (12/05/2021) en clase hemos visto (por encima)que se puede crear una clase y darle como atributos otra clase, esto abre un nuevo paradigma respecto a nuestro proyecto y cambia totalmente el modelo de clases que teníamos hasta ahora, aunque no se ha visto en profundidad (y no queda tiempo para realizar este cambio) paso a desarrollar la idea de como quedaría la nueva estructura de clases para el proyecto (**hay que hacer hincapié que es un planteamiento muy temprano y sin madurar la idea ya que carezco de varios conocimientos sobre esta manera de trabajar con las clases que se solventarán en futuras clases del ciclo**).

Primeramente tendríamos una clase llamada baraja o mazo, esta clase sería una clase que contendría otras clases como atributos, estas clases serían las de carta ya que el proyecto se puede extraer a la vida real como si fuera un mazo (que es el conjunto de todas las cartas en nuestro caso podría traducirse al thumbnail) y ese mazo dentro tiene cartas (que en nuestro caso cada atributo(objeto de carta) sería la carta en sí con su estructura y datos).

Como resumen a lo mencionado a lo anteriormente la anatomía de las clases quedaría de la siguiente manera:



```

@startuml
skinparam classAttributeIconSize 0

class Mazo {
    - class carta1
    - class carta2
    - class carta3
    - class carta4
    - class carta5
    + __construct($id_carta, $nombre_carta, $imagen_carta, $tipo)::cartas
    + mostrarthumbnails()
}

class Carta {
    - int $id_carta
    - string $nombre
    - string $elemento
    - string $iconomana
    - string $fondo
    - int $nummanacolor
    - int $manaincoloro
    - string $imgcriatura
    - string $tipo
    - string $subtipo
    - string $simbolexp
}

```

```

    - string $habilidad
    - string $descripcion
    - int|string $fuerza
    - int|string $resistencia
    - string $autor
    + __construct($id_carta, $nombre_carta, $color_elemento, $icono_mana, $fondo_carta, $num_mana_color, $num_mana_incoloro, $imagen_c
    + mostrarcarta()
}
Mazo --> Carta
@enduml

```

Creación de Base de Datos

Llegados a este punto teniendo clara la anatomía y estructura de datos que debemos guardar para mostrarlos posteriormente vamos a crear la Base de Datos.

Fijándonos en los atributos que tiene que tener la carta vemos de un vistazo inicial que una carta tiene un nombre, elemento al que pertenece, iconos mana de colores, un fondo de color, números de iconos de mana de color, número de mana incoloro, imagen de la criatura/artefacto, tipo, subtipo, ícono símbolo de la expansión, texto con la habilidad, texto con la descripción, fuerza, resistencia y autor.

Nosotros vamos a seguir una pauta muy sencilla en la que cada carta va a ser única es decir vamos a tener una serie de cartas por defecto con un identificador único, en el caso que haya una carta repetida no vamos a utilizar la carta ya existente y cambiar el valor de cartas que tenemos, en nuestro caso vamos a tratar cada carta como si fuera diferente (con un id único a cada una) aunque haya varias cartas con la misma información. Esta decisión la hemos tomado porque en realidad en un mazo de cartas físico si tenemos dos o más cartas iguales aunque tengan la misma información siguen siendo cartas independientes o dicho con otro ejemplo, unos hermanos gemelos son idénticos el uno al otro pero cada uno existe por si mismo y no tiene relación con el otro. En el caso de las cartas pasa exactamente igual ya que aunque tengan la misma información puede que una carta sea una reedición y cambie mínimamente la imagen o el tono de color del fondo ... por lo que aunque tengan la misma información las trataremos como diferentes (esta decisión guarda sentido y lógica a la hora de mostrar las cartas en pantalla).

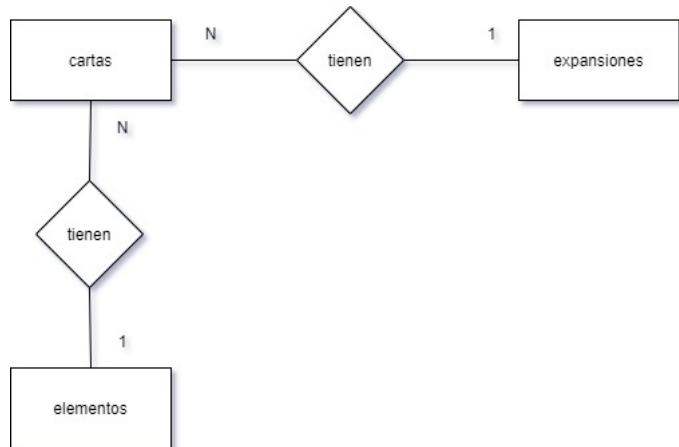
Modelo Conceptual

Ahora vamos a pasar a crear las relaciones, por lo que hemos visto tiene sentido tener la información de las cartas que va a ir cambiando en una tabla (aunque se repita información ya que aunque este sea el caso va a seguir siendo única y diferente), y por otro lado nos encontramos 2 atributos de la clase que tienen sentido crearlas como entidades, por lo cual vamos a tener 3 entidades :

- Las cartas: Datos de las cartas.
- Las expansiones: Datos de las expansiones (nombre e ícono).
- Los elementos: Datos de los elementos, que ícono de mana pertenece a cada uno y el fondo de la carta ya que todo va a ser del mismo color que el elemento.

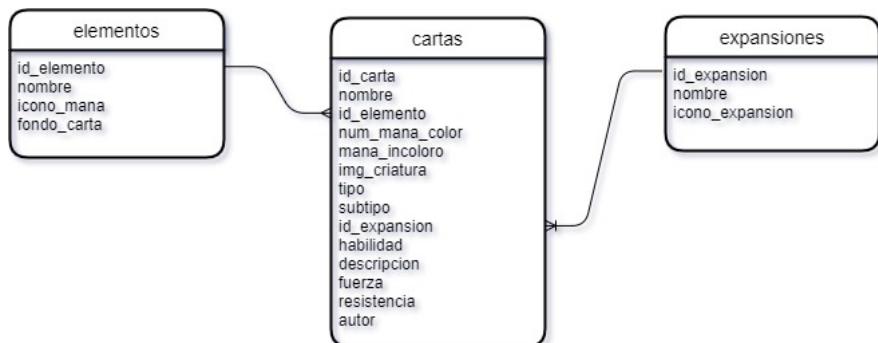
1 carta tiene 1 expansión y una expansión pueden tenerla varias cartas.

1 carta tiene 1 elemento y un elemento pueden tenerlo varias cartas.



Modelo lógico

Como hemos comentado anteriormente y una vez ya tenemos las relaciones claras que tienen nuestras entidades vamos a pasar a asignarle que atributos va a tener cada entidad.



Modelo físico

Una vez definido el modelo lógico es hora de asignar el tipo de datos que van a ser los atributos de cada entidad.



Script.sql de la Base de Datos

```
DROP DATABASE IF EXISTS proyecto_102;
CREATE DATABASE proyecto_102;
USE proyecto_102;
```

```

CREATE TABLE expansiones(
    id_expansion INT UNSIGNED AUTO_INCREMENT
    ,nombre VARCHAR(255)
    ,icono_expansion VARCHAR(1000)
    ,PRIMARY KEY (id_expansion)
);

CREATE TABLE elementos(
    id_elemento INT UNSIGNED AUTO_INCREMENT
    ,nombre VARCHAR(255)
    ,icono_mana VARCHAR(1000)
    ,fondo_carta VARCHAR(1000)
    ,PRIMARY KEY (id_elemento)
);

CREATE TABLE cartas(
    id_carta INT UNSIGNED AUTO_INCREMENT
    ,nombre VARCHAR(255)
    ,id_elemento INT UNSIGNED
    ,num_mana_color TINYINT UNSIGNED
    ,mana_incoloro TINYINT UNSIGNED DEFAULT 0
    ,img_criatura VARCHAR(1000)
    ,tipo VARCHAR(255)
    ,subtipo VARCHAR(255)
    ,id_expansion INT UNSIGNED
    ,habilidad VARCHAR(200)
    ,descripcion VARCHAR(200)
    ,fuerza VARCHAR(10)
    ,resistencia VARCHAR(10)
    ,autor VARCHAR(255)
    ,PRIMARY KEY (id_carta)
    ,FOREIGN KEY (id_expansion) REFERENCES expansiones(id_expansion)
    ,FOREIGN KEY (id_elemento) REFERENCES elementos(id_elemento)
);

insert into elementos (`nombre`, `icono_mana`, `fondo_carta`) values
('Azul','./img/elementos/mana/manaaazul.png', './img/elementos/fondos/fondoazul.jpg')
,('Blanco','./img/elementos/mana/manablanco.png', './img/elementos/fondos/fondoblanco.jpg')
,('Rojo','./img/elementos/mana/manarojo.png', './img/elementos/fondos/fondorojo.jpg')
,('Negro','./img/elementos/mana/mananegro.png', './img/elementos/fondos/fondonegro.jpg')
,('Verde','./img/elementos/mana/manaverde.png', './img/elementos/fondos/fondoverde.jpg');

insert into expansiones (`nombre`, `icono_expansion`) values
('Throne of Eldraine','./img/expansiones/throneofeldraine.png'),
('Commander 2019','./img/expansiones/commander.png'),
('Horizontes de Modern','./img/expansiones/modernhorizons.png'),
('La Guerra de la Chispa','./img/expansiones/warofspark.png');

insert into cartas (`nombre`, `id_elemento`, `mana_incoloro`, `num_mana_color`,
    ,`img_criatura`, `tipo`, `subtipo`, `id_expansion`, `habilidad`, `descripcion`, `fuerza`, `resistencia`, `autor`)
values
('Arconte de la Absolución',2,3,1,'./img/criaturas/01.png','Criatura','Arconte',1,'Vuela. Protección contra blanco.' 
    ,Esta criatura no puede ser bloqueada,hecha objetivo,recibirdrálo,estará encantada o ser equipada por nada blanco.)
,'Las criaturas no pueden atacarte a ti o a un planeswalker que controlas a menos que su controlador pague 1 mana incoloro por cada de esas criaturas.',3,2,'Igor Kierylek')
,('Paladín de Monteascua',3,3,1,'./img/criaturas/02.png','Criatura','Caballero humano',1,'Prisa. Tesón - Si se usaron al menos tres manás rojos para lanzar este hechizo,el Paladín de Monteascua entra al campo de batalla con un contador +1/+1 sobre él."El rey moriría por su reino.Yo no dudaría en hacer lo mismo por él"',4,1,'Randy Vargas')
,('Tortuga de Vadonublado',1,3,1,'./img/criaturas/03.png','Criatura','Tortuga',1,'Siempre que la Tortuga de Vadonublado ataque,otra criatura objetivo que no sea Humano no puede ser bloqueada este turno', 'Las Hadas criaron a la tortuga desde que salió del huevo.Le encanta transportar a través del río... y a quiénes ahogar.',1,5,'Mielvlos Ceran')
,('Filoscua',3,4,2,'./img/criaturas/04.png','Artefacto legendario','Equipo',1,'Destello Te cuesta 1 de mana incoloro lanzar este hechizo por cada criatura atacante que controlas. Cuando Filoscua entre al campo de batalla,anéxalo a la criatura objetivo que controlas. La criatura equipada obtiene +1/+1 y tiene las habilidades de dañar dos veces y arrollar. Equipar 3 de mana incoloro y/o encantamiento que controlas.', 'La alegría de un hada al conseguir su tesoro se desvanece rápidamente para convertirse en satisfacción,después aburrimiento y por último en la necesidad de robar otro poquito más.',1,'Iain McCaig')
,('Contendiente aclamada',2,2,1,'./img/criaturas/06.png','Criatura','Caballero humano',1,'Cuando la Contendiente aclamada entre al campo de batalla,si controlas otro Caballero,mira las cinco primeras cartas de tu biblioteca.Puedes mostrar una carta de canallero,Aura,Equipo que se encuentre entre ellas y ponerla en tu mano,Pon el resto en el fondo de tu biblioteca en un orden aleatorio.',3,3,'David')
,('Paladín del Valle de Arden',2,3,1,'./img/criaturas/07.png','Criatura','Caballero humano',1,'Tesón - Si se usaron al menos tres manás blancos para lanzar este hechizo, la Paladín del Valle de Arden entra al campo de batalla con un contador +1/+1 sobre ella.', 'Aun cuando la esperanza no es más que un candil que tilila en la noche,seguiré sirviendo al rey ausente.',2,5,'Volkan Baga')
,('Chiquitín',1,0,1,'./img/criaturas/08.png','Encantamiento','Aura',1,'Destello. Encantar criatura. La criatura encantada obtiene -2/-siete o más cartas en su cementerio,en vez de eso, obtiene -6/-0.', 'Su espada sonó como un tintineo argénteo contra el cristal del tarro y la duende sonrió encantada.',1,'Randy Vargas')
,('Syr Elenora,la Perspicaz',1,3,2,'./img/criaturas/09.png','Criatura legendaria','Caballero humano',1,'La fuerza de Syr Elenora,la Peña a la cantidad de cartas en tu mano. Cuando Syr Elenora entre al campo de batalla,roba una carta. A tus oponentes les cuesta 2 de mana lanzar hechizos que hagan objetivo a Syr Elenora.',1,4,'Mila Pesec')
,('Ayara la Primera de Nimboscuro',4,0,3,'./img/criaturas/10.png','Criatura legendaria','Noble elfo',1,'Siempre que Ayara, la Primera de Nimboscuro u otra criatura negra entre al campo de batalla bajo tu control,cada oponente pierde 1 vida y tú ganas 1 vida. Sacrificar otra criatura negra:Roba una carta.', 'El luto se torna en celebración en cuanto elige a su próximo pretendiente',2,3,'R

```

Diseño estructura

index

Ahora que tenemos la clase y la base de datos vamos a plantear la estructura de la página web. vamos a crear un index.php el cual va a albergar la estructura básica de la web, es decir va a contener la siguiente estructura:

```
<!DOCTYPE html>
<html lang="es">
  <head>
  </head>
  <body>
    <header>
      <nav></nav>
    </header>
    <main class="galeria">
    </main>
    <footer>
    </footer>
  </body>
</html>
```

El head va a contener el título de la página, metadatos y estilos.

El body va a contener:

- Header (cabecera de la web)
 - Menú de navegación (Inicio e insertar carta).
- Main (contenedor principal donde se va a encontrar nuestra galería (Cartas y thumbnails)).
- Footer (Pie de página que contiene opciones de ordenación e información de la baraja (número total de cartas y cuantas cartas hay repetidas)).

Ahora que tenemos la estructura principal vamos a desarrollar la estructura que va a tener la galería en nuestro main.

```
<main class="galeria">
  <div class="magic" data-nombre="" data-tipo="" data-default="">
    <figure class="thumbnail">
      <img>
      <figcaption></figcaption>
    </figure>
    <section class="carta" style=background-image:url()>
      <article class="header">
        <h2 class="nombre-carta"></h2>
        <div class="mana">
          <span class="mana-incoloro"></span>
          <div class="mana-color">
            <img class="icono-mana-color">
          </div>
        </div>
      </article>
      <article class="main">
        <div class="imagen-carta" style=background-image:url()>
        </div>
        <div class="tipo-carta">
          <h3></h3><img>
        </div>
        <div class="contenido-carta">
          <p class="efecto-carta"></p>
          <p class="descripcion-carta"></p>
        </div>
      </article>
      <article class="footer">
        <p class="autor-carta"></p>
        <div class="fuerzas">
          <i></i>
        </div>
      </article>
    </section>
  </div>
</main>
```

Como se puede observar nuestro main se diferencia en 3 partes:

Magic: (Contenedor que contiene el thumbnail y la carta)

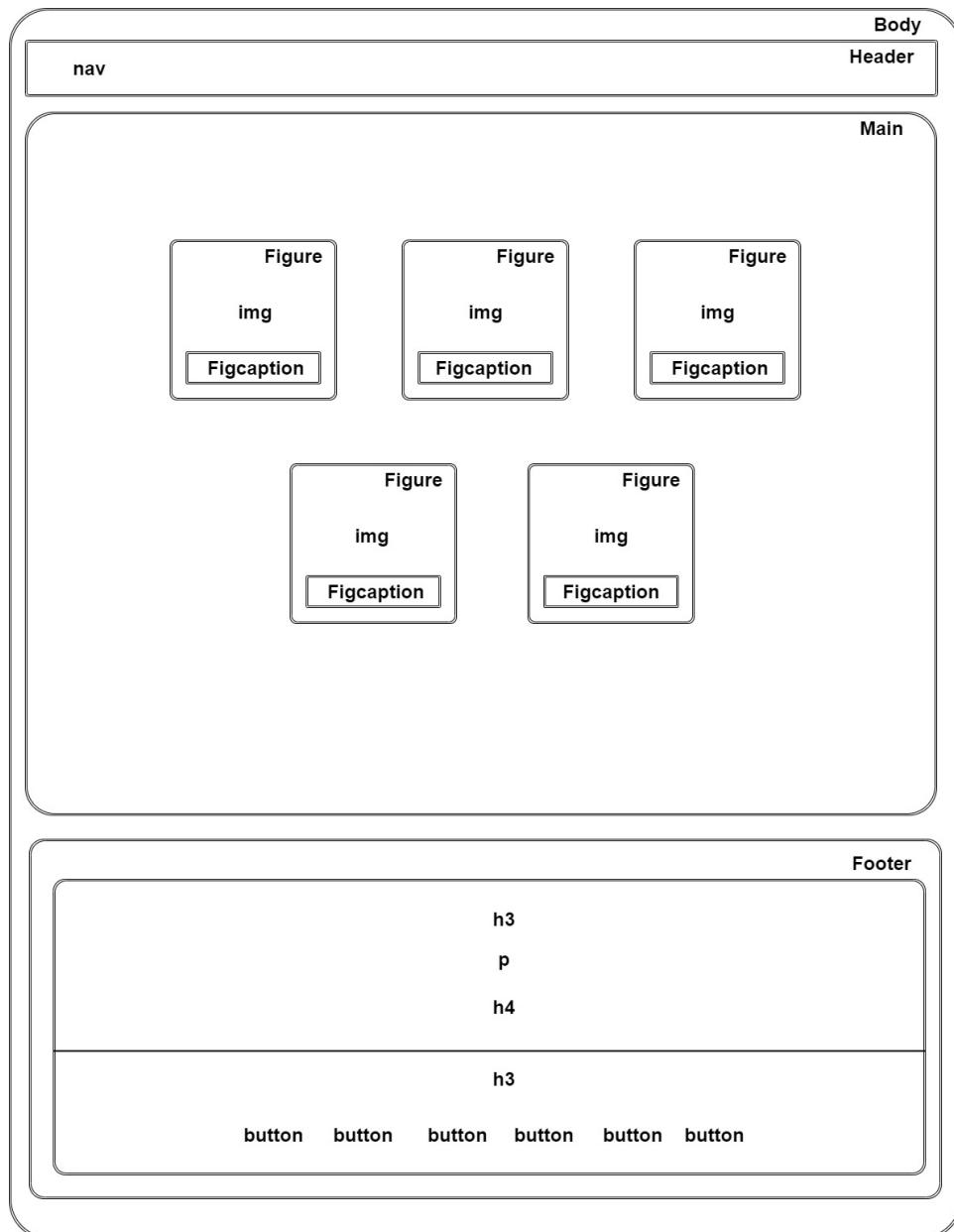
Thumbnail: (Imagen y texto de previsualización de la carta).

Carta: (Anatomía completa de la carta a tamaño real y sus proporciones).

El thumbnail es una etiqueta figure que contiene:

La imagen de la criatura/artefacto.

El figcaption que va a contener el nombre de la carta.

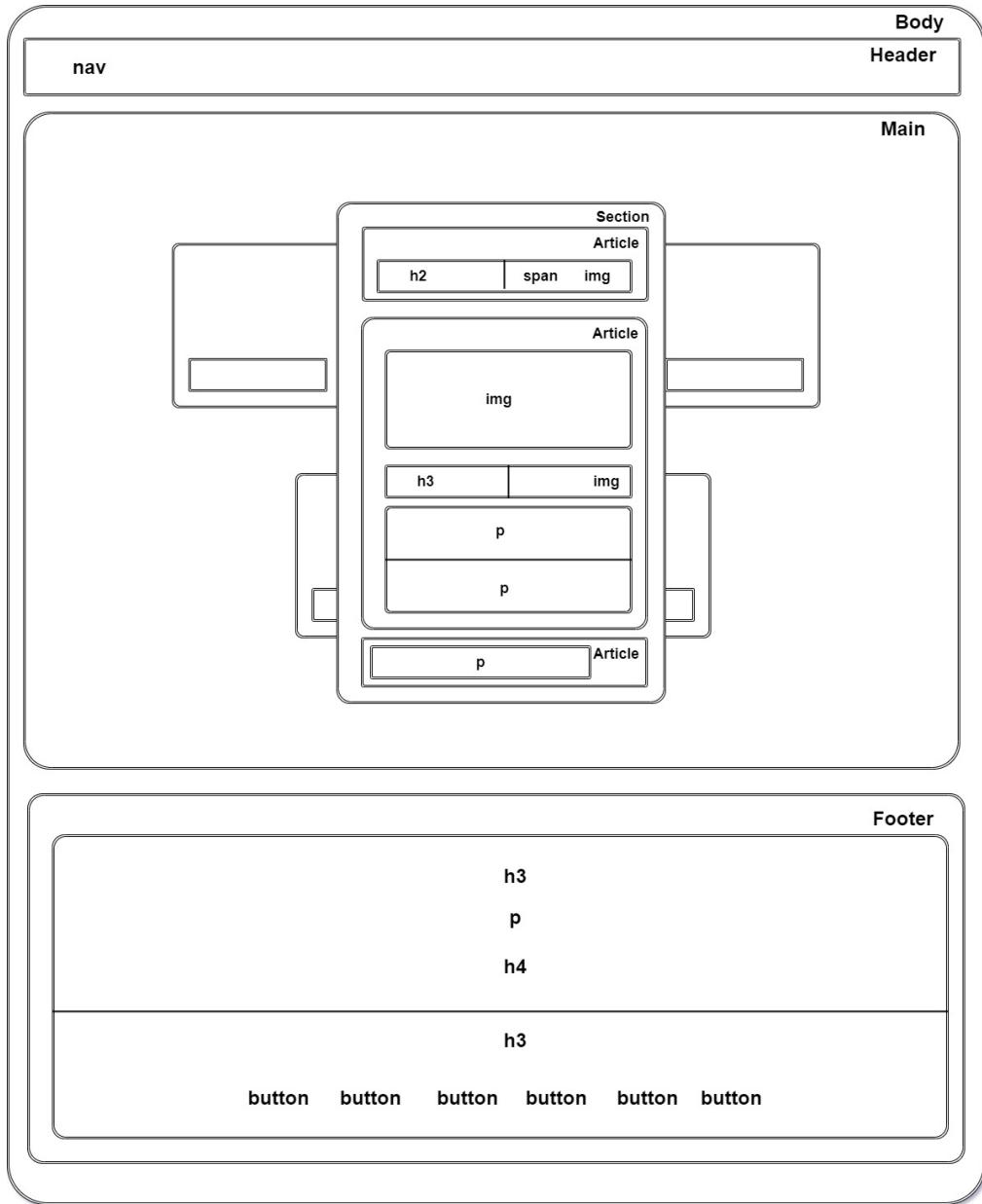


La carta es un contenedor, este contenedor contiene un section (contenedor general de la carta) y tres article (header,main y footer de la carta) estos 3 article contienen:

Header: (Cabecera de la carta (nombre, mana incoloro e iconos de mana de color)).

Main: (Contiene la imagen de la carta/artefacto, el tipo, subtipo e icono de expansión a la que pertenece y la descripción de la carta.(*Hay que aclarar que en una web solo puede existir un main por lo tanto en este caso tenemos que ponerlo como la clase del article*))

Footer:(Contiene el pie de página de la carta es decir los puntos de fuerza / resistencia y el nombre del autor).



CSS

Nuestra aplicación tiene que ser adaptable para todo tipo de pantallas, esto quiere decir que nos vamos a centrar en realizar una aplicación validada en RWD (Responsive Web Design). A medida que hemos diseñado nuestra estructura anterior en HTML5 hemos ido pensando y adaptando esta estructura para poder aplicar una serie de estilos y técnicas en CSS para que casi por arte de magia nuestra aplicación sea RWD.

Hemos utilizado varias técnicas pero las más utilizadas y relevantes son el position absolute, relative, fixed y flexbox. Los position los hemos utilizado para superponer la fuerza y resistencia de la carta sobre la misma y para fijar la carta modal en el centro de la pantalla y sobreponerla a los thumbnail.

Aunque estas técnicas son muy importantes para el posicionamiento de la carta y la ventana modal sobre todo el contenido de la aplicación la verdadera técnica estrella es FLEXBOX, esta técnica es la verdadera autora de que nuestra web sea RWD teniendo como aliados a los media query.

Para utilizar flexbox nos hemos apoyado en una guía escrita por un miembro oficial que pertenece a la mesa de decisiones de la W3C.

A Complete Guide to Flexbox | CSS-Tricks

Our comprehensive guide to CSS flexbox layout. This complete guide explains everything about flexbox, focusing on all the different possible properties for the parent element (the flex container) and the child elements (the flex items). It also includes history, demos, patterns, and a browser support chart.

* <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



Javascript

Relacionando thumbnails con cartas

Como hemos podido ver una carta se imprime en pantalla cuando existe su thumbnail y viceversa, estos tienen un parentesco de hermanos y un padre llamado magic, para poder mostrar la carta correcta al hacer click en un thumbnail le añadimos un evento click a cada thumbnail, cuando este evento ocurra nos vamos a dirigir al padre de ese thumbnail que en este caso va a ser siempre un contenedor llamado magic. Una vez nos encontramos en el padre del thumbnail al que le hemos dado click vamos a pasar a buscar a su hermano pequeño , en este caso cogeremos el segundo hijo del padre magic que siempre según la estructura que hemos diseñado anteriormente va a ser la carta que queremos mostrar, al estar relacionadas con parentesco no vamos a tener problemas de incongruencias de datos a la hora de hacer click en un thumbnail y que nos muestre otra cosa que no sea la carta hermana de ese thumbnail.

```
thumbnails.forEach(function (thumbnail) {
    thumbnail.addEventListener('click', function () {
        removeActivaClase();
        this.parentNode.children[1].classList.add('activa');
        this.parentNode.children[1].scrollIntoView({block: "end", behavior: "smooth"});
    })
})
```

+

Ocultar la carta

Para ocultar la carta y ver todo el thumbnail completo sin la carta de la ventana modal en medio vamos funciones para que al hacer click en la carta desaparezca y poder continuar navegando por nuestra aplicación.

Primero vamos a crear una función que va a ocultar todas las cartas ya que no queremos que se muestre ninguna.

```
cartas.forEach(function (carta) {
    carta.addEventListener('click', function () {
        removeActivaClase();
    })
})
```

Segundo vamos a añadirle a todas las cartas un evento click el cual cuando hagan click en una carta llamará a la función anterior y se ocultará, las demás cartas al tener display none no van a mostrarse en la pantalla por lo que no se les puede hacer click y evitar problemas de funcionamiento.

```
cartas.forEach(function (carta) {
    carta.addEventListener('click', function () {
        removeActivaClase();
    })
})
```

Ordenando nuestras cartas

Para ordenar las cartas hemos creado unos botones en el footer para ordenar de manera ASC/DESC por nombre, tipo y orden por defecto que la base de datos. Primero hemos creado una lista de nodos utilizando una técnica para establecer los atributos que queremos en nuestras etiquetas llamado dataset. Como hemos realizado estos pasos? Muy sencillo, como hemos comentado en la estructura HTML5 de la carta vemos que el contenedor magic tiene unos atributos que comienzan con data, estos es lo que llamamos dataset.

```
<div class="magic" data-nombre="nombre_carta" data-tipo="tipo_carta" data-default="id_default">
```

Cada carta magic tiene 3 dataset uno en el que se guarda el nombre de la carta, otro en el que se guarda el tipo de la carta y un último atributo que guarda el orden por defecto que llega de la base de datos.

```
magics.forEach(function (magic) {
    arnombres = document.querySelectorAll(".magic")[contador].dataset.nombre;
    artipos = document.querySelectorAll(".magic")[contador].dataset.tipo;
    ardefault = document.querySelectorAll(".magic")[contador].dataset.default;
    nombres.push(arnombres);
    tipos.push(artipos);
    pordefecto.push(ardefault);
    contador++;
})
```

Una vez creados las listas de nodos con los valores de los dataset que vamos a ordenar pasamos a crear tres funciones, estas funciones cuando se le da click al botón de ordenar ASC van a coger la lista de nodos creada y le van a realizar un .sort() para ordenarlos ASC, una vez tenemos la lista de nodos ordenada como queremos vamos a tener que ir comparando la lista de nodos que nos crea por defecto el querySelectorAll() y la lista de nodos que hemos creado con sus respectivos valores. El funcionamiento es el siguiente:

1. Compara el primer valor de la lista de nodos ordenada con el primer valor de la lista de nodos por defecto.
2. Si no son iguales compara el primer valor de la lista de nodos ordenada con el segundo valor de la lista de nodos por defecto.
3. Realiza este proceso sucesivamente hasta encontrar el valor que hace match entre uno y otro (Siempre lo va a encontrar ya que el ordenado se crea a partir de el por defecto.(Si llega al final y no lo encuentra vuelve a comparar el array por defecto hasta encontrarlo, si por algún casual no lo encuentra pasaría al siguiente valor de la lista ordenada)).
4. Una vez hacen match los valores se le asigna a ese contenedor magic un style con un order (propiedad de flexbox que permite ordenar elementos flex) igual al valor del índice de la lista de nodos al que hace match, de esta manera nunca habrá un order repetido ya que los índices son consecutivos y no se repiten y al estar ordenados con sort nos aseguramos que los índices contengan valores ordenados.

Para realizar el orden DESC se aplicaría la misma lógica solamente que primero se realizaría un sort() y posteriormente un reverse() para que los ordene DESC.

```
function añadirstyleordernombre() {

    nombres.forEach(function (elemento, indice, nombres) {
        contador = 0;
        magics.forEach(function (magic) {
            if (elemento == document.querySelectorAll(".magic")[contador].dataset.nombre) {
                document.querySelectorAll(".magic")[contador].style.order = indice;
            }
            contador++;
        })
        ordenultimoelementos();
    })

    ordennombreasc.addEventListener('click', function () {
        eliminarorder()
        nombres.sort();
        añadirstyleordernombre();
    });

    ordennombredesc.addEventListener('click', function () {
        eliminarorder()
        nombres.sort();
        nombres.reverse();
        añadirstyleordernombre();
    })
}
```

Además necesitamos añadir una función para evitar que el orden pueda generarse de manera errónea, es decir que cuando se haga click en un botón de ordenar se reinicie el valor de todos los order de las magic a ninguno.

```
function eliminarorder() {
    magics.forEach(function (magic) {
        magic.style.order = "";
    })
}
```

Información de la carta

Para los dos requisitos de mostrar la cantidad total de cartas que tenemos y mostrar cuantas cartas tenemos de cada hemos optado por la creación toggle muy simple el cual va a tener un título indicando que contiene la información de las cartas y unos iconos de flechas para abajo indicando al usuario que al pincharle se muestra más contenido abajo, se ha creado de esta manera ya que si se dejaba la información fija en pantalla primero que la experiencia de usuario no era muy buena ya que siempre tenia información en la pantalla que no quería en ese momento y por que esa información fija en pantalla era demasiado extensa sobre todo en dispositivos móviles.

```
informacionh3.addEventListener('click', function () {
    if(informacioncarta[0].classList.contains("informacioncarta")){
        informacioncarta[0].classList.remove('informacioncarta');
        informacioncarta[0].scrollIntoView({block: "end", behavior: "smooth"});
    }
    else{
        informacioncarta[0].classList.add('informacioncarta');
        informacionh3.scrollIntoView({block: "end", behavior: "smooth"});
    }
})
```

Formulario

Estructura

La estructura del formulario va a ser la siguiente:

```
function añadircarta()
{
    require_once ("conexionbd.php");
    echo "
        <form action=? method=POST enctype=multipart/form-data>
            <h2>¿Deseas añadir una carta?</h2>
            <label for=nombre>Nombre:</label><br>
            <input name=nombre id=nombre type=text><br>
            <label for=incoloro>Número Mana Incoloro:</label><br>
            <input name=incoloro id=incoloro type=number min=0 max=9 step=1></input><br>
            <label for=color>Elemento:</label><br>
            <select name=color id=color>
                <option value=1>Azul</option>
                <option value=2>Blanco</option>
                <option value=3>Rojo</option>
                <option value=4>Negro</option>
                <option value=5>Verde</option>
            </select><br>
            <label for=manacolor>Número Mana Color:</label><br>
            <input name=manacolor id=manacolor type=number min=0 max=3 step=1></input><br>
            <label for=archivo>Añadir imagen:</label><br>
            <input name=archivo id=archivo type=file><br>
            <label for=tipo>Tipo:</label><br>
            <input name=tipo id=tipo type=text></input><br>
            <label for=subtipo>Subtipo:</label><br>
            <input name=subtipo id=subtipo type=text></input><br>
            <label for=expansion>Expansión:</label><br>
            <select name=expansion id=expansion>
                <option value=1>Throne of Eldraine</option>
                <option value=2>Commander 2019</option>
                <option value=3>Horizontes de Modern</option>
                <option value=4>La Guerra de la Chispa</option>
            </select><br><br>
            <label for=habilidad>Habilidad:</label><br>
            <textarea style=resize:none id=habilidad name=habilidad rows=4 cols=30></textarea><br>
            <label for=descripcion>Descripción:</label><br>
            <textarea style=resize:none id=descripcion name=descripcion rows=4 cols=30></textarea><br>
            <label for=fuerza>Número Fuerza:</label><br>
            <input name=fuerza id=fuerza type=text></input><br>
```

```

<label for=resistencia>Número Resistencia:</label><br>
<input name=resistencia id=resistencia type=text></input><br>
<label for=autor>Autor:</label><br>
<input name=autor id=autor type=text></input><br>
<input type=submit id=enviar name=enviar value='Crear&#32;Carta'></input><br>
</form>
";

```

Tipos de datos y validaciones

- El nombre es de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena y no supere una longitud de 1 a 27 caracteres.
- El número de mana incoloro es de tipo entero y se va a comprobar que el número antes de insertarlo en la base de datos no sea menor que 0 ni mayor que 10.
- El elemento es de tipo entero y se va a comprobar que el número recibido se encuentre entre el rango de 1 y 5 incluidos.
- El número de mana de color es de tipo entero y se va a comprobar que el número recibido se encuentre entre el rango 0-3 incluidos.
- La carta de la imagen va a ser de tipo string para poder tratar la ruta donde se sube la imagen. Se va a comprobar que tenga una extensión válida y que el archivo no exista ya en el directorio donde se va a subir.
- El tipo va a ser de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena, longitud máxima de 50 caracteres.
- El subtipo va a ser de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena, longitud máxima de 50 caracteres.
- El símbolo de la expansión es de tipo entero y se va a comprobar que el número recibido se encuentre entre el rango de 1 y 4 incluidos.
- La habilidad va a ser de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena y que no supere una longitud máxima de 100 caracteres.
- La descripción va a ser de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena y que no supere una longitud máxima de 100 caracteres.
- La fuerza va a ser de tipo entero o string y la validación que se va a realizar es que solo tenga o un número o un carácter, si se dan valores numéricos se encuentren en un rango del 0 al 9 incluidos y si es un string que solo pueda ser el carácter *.
- La resistencia va a ser de tipo entero o string y la validación que se va a realizar es que solo tenga o un número o un carácter, si se dan valores numéricos se encuentren en un rango del 0 al 9 incluidos y si es un string que solo pueda ser el carácter *.
- El autor es de tipo string y la validación que se le va a realizar es que antes de insertarlo en la base de datos se le va a hacer un trim para evitar espacios tanto al principio como al final de la cadena y no supere una longitud máxima de 50 caracteres.

Batería de pruebas

La estrategia que se ha seguido es primero crear funciones en la clase para que sea capaz de auto verificar si los datos con los que se va a construir el objeto son correctos y pasan todos los escenarios previstos. Para ello hemos usado las expresiones regulares PCRE de preg_match después hemos ido creando los test y verificando si se pasaban estas comprobaciones con estos datos, en los preg_match de texto primero hemos ido comprobando mayúsculas, minúsculas y números y a medida que íbamos realizando pruebas nos hemos dado cuenta que había que mejorar la expresión regular ya que no acertaba caracteres, acentos ñ ... Al final una vez realizadas muchas pruebas hemos llegado a la conclusión de utilizar la opción ascii y cuando solo queríamos letras la alpha además de añadir los acentos y letra ñ en sus variantes, de esta forma y gracias a los test hemos detectado muchos bugs en nuestros primeros preg_match y hemos podido solventarlos.

Ejemplo

```

/**
 * Comprueba si el nombre es válido y se crea correctamente
 * @return true Devuelve true si se ha creado correctamente y era válido
 * @return false Devuelve false es válido
 */

public function comprobarnombre($nombre_carta){
    if(preg_match("/^[:ascii:]\áéíóúÁÉÍÓÚñ]{1,27}$/", $nombre_carta)){
        return true;
    }
    return false;
}

/**
 * @covers ::comprobarnombre()
 */

public function testElNombreEstaFormadoCorrectamenteYSeCrea(){
    //Given
    $id_carta = "11";
    $nombre_carta = "Nombre Carta camión";
    $color_elemento = "Blanco";
    $icono_mana = "./img/elementos/manas/manablanco.png";
    $fondo_carta = './img/elementos/fondos/fondoblanco.jpg';
    $num_mana_color = 2;
    $num_mana_incoloro = 4;
    $imagen_carta = './img/criaturas/01.png';
    $tipo = "Tipo";
    $subtipo = "Subtipo";
    $icono_expansion = 1;
    $habilidad = "Aqui va la habilidad";
    $descripcion = "Aqui va la descripcion";
    $fuerza = "*";
    $resistencia = 2;
    $autor = "Ismael Aliaga";
    $carta = new Carta($id_carta, $nombre_carta, $color_elemento, $icono_mana, $fondo_carta, $num_mana_color, $num_mana_incoloro,
    $habilidad, $descripcion, $fuerza, $resistencia, $autor);
    //When
    $resultado = $carta->comprobarnombre($nombre_carta);
    //Then
    $this->assertTrue($resultado);
}

/**
 * @covers ::comprobarnombre()
 */

public function testElNombreNoEstaFormadoCorrectamenteYNoSeCrea(){
    //Given
    $id_carta = "11";
    $nombre_carta = "Nombre Cartaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
    $color_elemento = "Blanco";
    $icono_mana = "./img/elementos/manas/manablanco.png";
    $fondo_carta = './img/elementos/fondos/fondoblanco.jpg';
    $num_mana_color = 2;
    $num_mana_incoloro = 4;
    $imagen_carta = './img/criaturas/01.png';
    $tipo = "Tipo";
    $subtipo = "Subtipo";
    $icono_expansion = 1;
    $habilidad = "Aqui va la habilidad";
    $descripcion = "Aqui va la descripcion";
    $fuerza = "*";
    $resistencia = 2;
    $autor = "Ismael Aliaga";
    $carta = new Carta($id_carta, $nombre_carta, $color_elemento, $icono_mana, $fondo_carta, $num_mana_color, $num_mana_incoloro,
    $habilidad, $descripcion, $fuerza, $resistencia, $autor);
    //When
    $resultado = $carta->comprobarnombre($nombre_carta);
    //Then
    $this->assertNotTrue($resultado);
}

```

Herramientas utilizadas

PlantUML

PlantUML Web Server

Create simply and freely UML diagrams from your browser thanks to PlantUML Web Server. Just enter a text diagram, and get the result in PNG or SVG format.

🔗 <http://www.plantuml.com/plantuml/uml/SyffFKj2rKt3CoKnELR1Io4ZDoSa70000>



PHP Debug

xdebug/vscode-php-debug

Manage pull requests and conduct code reviews in your IDE with full source-tree context. Comment on any line, not just the diffs. Use jump-to-definition, your favorite keybindings, and code intelligence with more of your workflow. Learn More If you find this extension useful, if it helps you solve your problems and if

🔗 <https://github.com/xdebug/vscode-php-debug>

xdebug/vscode-php-debug



PHP Debug Adapter for Visual Studio Code

24 Contributors 125 Issues 584 Stars 134 Forks

Notion

Notion - The all-in-one workspace for your notes, tasks, wikis, and databases.

A new tool that blends your everyday work apps into one. It's the all-in-one workspace for you and your team.

🔗 <https://www.notion.so/>



Notion

The all-in-one workspace.
Notes, tasks, wikis, & databases.



Draw.io

Flowchart Maker & Online Diagram Software

diagrams.net (formerly draw.io) is free online diagram software. You can use it as a flowchart maker, network diagram software, to create UML online, as an ER diagram tool, to design database schema, to build BPMN online, as a circuit diagram maker, and more. draw.io can import .vsdx, Gliffy™ and Lucidchart™ files .

🔗 <https://app.diagrams.net/>

Problemas y soluciones

En un principio nuestra estructura era muy diferente a la estructura comentada anteriormente. En esta antigua estructura imprimíamos los thumbnails por un lado y las cartas por otro lado, este sistema podía llegar a plantear alguna incongruencia de datos como por ejemplo, que se creara un thumbnail y no la carta o que se creara la carta y no el thumbnail. Otro problema que nos encontrábamos era a la hora de relacionar que carta pertenecía a que thumbnail a la hora de trabajar con JavaScript, después de pensar y comentar el problema con el cliente llegó a la conclusión que la mejor manera era agrupar los thumbnails y cartas todo en un mismo contenedor. De esta nueva forma solucionábamos todos los problemas que teníamos con la estructura anterior, ya que al crear un contenedor llamado magic que recogía el thumbnail y la carta nos asegurábamos de que al crear el thumbnail se creaba la carta y se solucionaba el problema para relacionar el thumbnail con la carta ya que estas tienen un parentesco de hermanos y comparten un mismo padre que es magic en este caso.

Bibliografía

A Complete Guide to Flexbox | CSS-Tricks

Our comprehensive guide to CSS flexbox layout. This complete guide explains everything about flexbox, focusing on all the different possible properties for the parent element (the flex container) and the child elements (the flex items). It also includes history, demos, patterns, and a browser support chart.

🔗 <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>



Referencia del lenguaje

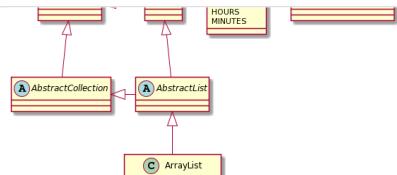
🔗 <https://www.php.net/manual/es/langref.php>



sintaxis diagrama de clases y características

Las relaciones entre clases se definen usando los siguientes símbolos: Type Symbol Drawing Extension . También puedes definir notas usando las palabras claves note left of , note right of , note top of , note bottom of . Además puedes definir una nota en la última clase definida usando note left , note right , note

🌐 <https://plantuml.com/es/class-diagram>



HTML 5.2

The English version of this specification is the only normative version. Non-normative translations may also be available. This specification defines the 5th major version, second minor revision of the core language of the World Wide Web: the Hypertext Markup Language (HTML).

🌐 <https://www.w3.org/TR/html52/>

MySQL 8.0 Reference Manual

This is the MySQL™ Reference Manual. It documents MySQL 8.0 through 8.0.27, as well as NDB Cluster releases based on version 8.0 of through 8.0.26-ndb-8.0.26, respectively. It may include documentation of features of MySQL versions that have not yet been released. For information about which versions have been released, see the MySQL 8.0 Release Notes.

🌐 <https://dev.mysql.com/doc/refman/8.0/en/>

HTMLElement.dataset - Referencia de la API Web | MDN

La propiedad dataset en HTMLElement proporciona una interfaz lectura/escritura para obtener todos los atributos de datos personalizados (data-*) de cada uno de los elementos. Está disponible el acceso en HTML y en el DOM. Dentro del map de DOMString, aparece una entrada por cada atributo de datos.

🔗 <https://developer.mozilla.org/es/docs/Web/API/HTMLElement/dataset>



Referencia de Tecnologías Web | MDN

La Web abierta (open Web) se construye usando diversas tecnologías. A continuación encontrará enlaces a nuestro material de referencia para cada una de ellas. El Lenguaje de Marcado para Hipertextos (HyperText Markup Language) es usado para describir y definir el contenido de una página Web en un formato bien

🔗 <https://developer.mozilla.org/es/docs/Web/Reference>

